TECHNOLOGIES FOR INFORMATION SYSTEMS

Prof. Letizia Tanca
Author: Davide Giannubilo – a.y. 2023/24

1. Introduction

The four V's of Big Data:

1. Volume (huge volume of data)
2. Velocity (rate)
3. Variety (data sources, even in the same domain are extremely heterogeneous both at the schema level, regarding how they structure their data, and at the instance level, regarding how they describe the same real-world entity, exhibiting considerable variety even for substantially similar entities)
4. Veracity (data sources, even in the same domain, are of widely differing qualities, with significant differences in the coverage, accuracy and timeliness of data provided)

2. Introduction to (big) data integration

The actual implementation of the Data Analysis (ML, statistics, Data Mining…) algorithm is usually less than 5% lines of code in a real, non-trivial application. The main effort (i.e. those 95% LOC) is spent on:

- Data Cleaning e annotation
- Data extraction, transformation, loading
- Data Integration e pruning
- Model training e deployment

The **data integration problem** is composed of two parts:

1. combine data coming from different data sources and
2. provide to the user a unified view of the data.

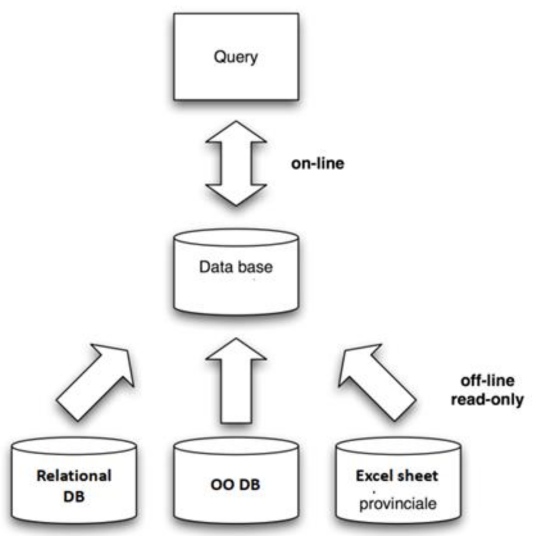
The **data integration process** is done in three steps:

1. Schema reconciliation → mapping the data structure
2. Record linkage (aka Entity resolution) → data matching based on the same content
3. Data fusion → reconciliation of non-identical content

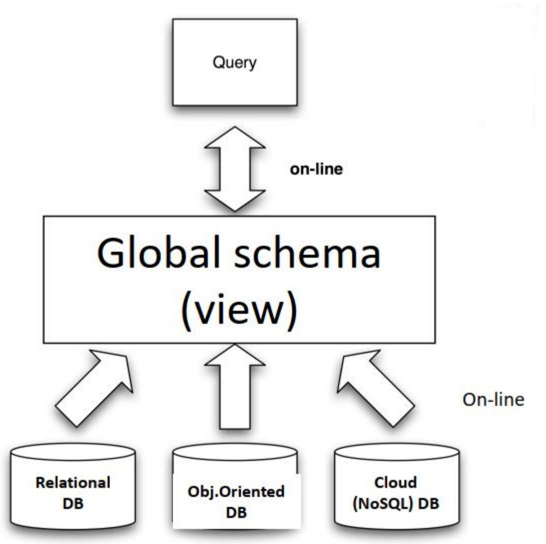
There two ways of integrating two database systems:

- Use a **materialized database** (data are merged in a new database) → Extract-Transform-Load (ETL) Systems
    - Data Warehouses: materialized integrated data sources
- Use a **virtual non-materialized database** (data remain at sources) → Enterprise Information Integration (EII) (or Data Integration) Systems (common front-end to the various data sources)
    - Data Exchange (source-to-target), less used recently

MATERIALIZED



VIRTUAL



Materialized integration
Large common stores came to be known as warehouses, and the software to access, scrape, transform, and load data into warehouses, became known as extract, transform, and load (ETL) systems.

In a dynamic environment, one must perform ETL periodically (say once a day or once a week), thereby building up a history of the enterprise.

The main purpose of a data warehouse is to allow systematic or ad-hoc data analysis and mining. Not appropriate when the need is to integrate the operational systems (keeping data up to date).

Virtual integration
The virtual integration approach leaves the information requested in the local sources. The virtual approach will always return a *fresh answer* to the query. The query posted to the global schema is reformulated into the formats of the local information system. The information retrieved needs to be combined to answer the query.

The conventional wisdom is to use data warehousing and ETL products to perform data integration. However, there is a **serious flaw** in one aspect.

Suppose one wants to integrate current (operational) data rather than historical information. Consider, for example, an ecommerce web site which wishes to sell hotel rooms over the web. The actual inventory of available hotel rooms exists in 100 or so information systems since Hilton, Hyatt and Marriott all run their own reservation systems.
Applying ETL and warehousing to this problem will create a copy of hotel availability data, which is quickly out of date. But, if a web site sells a hotel room, based on this data, it has no way of guaranteeing delivery of the room, because somebody else may have sold the room in the meantime. So, when you need the integrated data to be **always up to date**, we must use **VIRTUAL INTEGRATION**.

2.1 Query execution in the integrated system
When a query is submitted, the integration system has to decompose it into queries against the component datasets:

1. determine first which parts of the query refer to which dataset,
2. which parts apply to only a single dataset and
3. which parts apply to data from different datasets.

The latter ones can only be evaluated over the integrated data (view), whereas the former ones can be evaluated within the component datasets.

One of the main challenges in developing virtual data integration systems is to find good strategies of how to decompose and evaluate queries against multiple databases in an efficient manner.

Data integration problems arise even in the simplest situation when we have a unique and centralized DB because imagine that a global company DB is just a collection of some partial DBs, so there will be a lot of redundancies in those data and there could be inconsistencies if we update one instance and the other remain as before.
The correct answer is to use a single, integrated and centralized DB. This eliminates useless

redundancies which cause useless memory occupation and inconsistencies in case of updates.
Each functionality in the company will have its own personalized view of the central DB and this is achieved by using **view definitions**.

2.2 Schema reconciliation

Starting from, at least, two data sources that have the same data model, we have to create a *global schema* that will provide a *reconciled*, *integrated* and *virtual* view of the data.

1. Source schema identification (when present)
2. Source schema reverse engineering (data source conceptual schemata)
3. Conceptual schemata integration and restructuring: conflict resolution and restructuring
4. Conceptual to logical translation (of the obtained global schema)
5. Mapping between the global logical schema and the single schemata (logical view definition)
6. After integration: query-answering through data views

At the third point we have conflicts, and they could be:

- Related concept identification
    - ex. employee – clerk or exam – course
- Conflict analysis and resolution
    - Name conflicts, homonyms or synonims
    - Type conflicts
        - ex. in the attribute “gender” could be “M/F” or “Male/Female”
        - in the entity, different abstractions of the same real world concept produce different sets of attributes
    - Data semantics conflicts
        - Currencies
        - Measure systems
    - Structure conflicts
    - Dependency or cardinality conflicts
    - Primary key conflicts
- Conceptual schema integration

3. Structured data integration

In general, a data integration system is a triple (G, S, M).

In the integrated system, the query is posed in terms of G and specifies in which data of the virtual database we are interested. The problem is understanding which real data correspond to the virtual ones.

So, there is a mapping between the global logical schema and the sole source schemata (in a logical view of the system).
We have two basic techniques:

1. **GAV** (Global As View)
2. **LAV** (Local As View)

The same approach can be used also in case of different data models. In this case a model transformation is required.

3.1 GAV

In the GAV approach, we suppose that the global schema is derived from the integration process of the data source schemata, thus the global schema is expressed in terms of it.

A GAV mapping is a set of assertions, one for each element g of G: $g \rightarrow q_S$

The mapping specifies g as query q_S over the data sources. This means that the mapping tells us exactly how the element g is computed.
(This approach is OK for stable data source, so it is difficult to extend with a new data source)

SOURCE 1	GLOBAL SCHEMA

Product(Code, Name, Description, Warnings, Notes, CatID)
 	
Version(ProductCode, VersionCode, Size, Color, Name, Description, Stock, Price)
	CREATE VIEW GLOB-PROD AS
	SELECT Code AS PCode, VersionCode as VCode, Version.Name AS Name, Size, Color, Version.Description as Description, CatID, Price, Stock
	FROM SOURCE1.Product, SOURCE1.Version
	WHERE Code = ProductCode
	
	UNION
	
	SELECT Code AS PCode, null as VCode, Name, Size, Color, Description, Type as CatID, Price, Q.ty AS Stock
	FROM SOURCE2.Product

SOURCE 2	
Product(Code, Name, Size, Color,Description, Type, Price, Q.ty)	

Suppose we introduce a new source. The simple views we have just created must be modified.
In the simplest case (like in the GLOB_PROD case) we only need to add a union with a new SELECT-FROM-WHERE clause. This is not true in general; view definitions may be much more complex.

3.2 LAV

In the LAV approach, the global schema has been designed independently of the data source schemata. The relationship (mapping) between sources and global schema is obtained by defining each data source as a view over the global schema.

A mapping LAV is a set of assertions, one for each element s of each source S: $s \rightarrow q_G$

The content of each source is characterized in terms of a view q_G over the global schema.
(It is OK if the global schema is stable, it allows extensibility, but query processing is more complex than expected)

Queries are expressed in terms of the global schema; therefore the needed transformation is the inverse of the defined views: some *reasoning* is needed.
A **plan** is a rewriting of the query as a set of queries to the sources and a prescription of how to combine their answers.

3.3 LAV and GAV comparison

GAV	LAV
Mapping quality depends on how well we have compiled the sources into the global schema through the mapping	Mapping quality depends on how well we have characterized the sources
Whenever a source changes or a new one is added, the global schema needs to be reconsidered	High modularity and extensibility (if the global schema is well designed, when a source changes or is added, only its definition is to be updated)
	Query processing needs reasoning

3.4 GAV operators

The most useful integration operators are:

- Union
- Outerunion
- Join
- Outerjoin
- Generalization

3.5 Record Linkage and Data Fusion

Record Linkage

Also known as Entity Resolution, it aims to find the information that refer to same real-world entities.
Considering the case of a relational database (most common), several techniques can be used:

- String matching, concatenate multiple columns into a string, then compare the two strings. Not suggested!
- Tuple (or structured data) matching, compare records field by field. It is much easier to spot similarities since we can also associate different meanings to the columns.

- String similarity and similarity measures, given two sets of strings, find all pairs from the two sets that refer to the same real-world entity. Each of these pairs is called a match.
  - Types of similarity measures:
    - Sequence-based
      - edit-distance, based on the minimal number of operations that are needed to transform string *a* to string *b*
    - Set-based
      - Jaccard: divide the strings into tokens, and compute the measure on the two sets of tokens (intersection divided by union of tokens)
    - Phonetic
      - Soundex: calculates a four-character code from a word based on the pronunciation and considers two words as similar if their codes are equal. Similar sounding letters are assigned the same Soundex code. Note that such technique is strongly language based and may fail in some cases (for example, same pronunciation words...)
- Record matching
  - Rule-based matching, manually written rules that specify when two tuples match (e.g., two tuples refer to the same person if they have the same SSN)
  - Learning Matching Rules
    - Supervised, learn how to match from training data, then apply it to match new tuple pairs (requires a lot of training data)
    - Unsupervised, clustering records based on similar values
  - Probabilistic Matching, model the matching domain using a probability distribution. Provides a principled framework that can naturally incorporate a variety of domain knowledge. It is computationally expensive and hard to understand fully.

Data Fusion: once recognized that two items refer to the same entity, how do we reconcile inconsistent information?

These inconsistencies may depend on different reasons like one or both of the sources are incorrect, or each source has a correct but partial view.

3.6 Wrappers

Wrapper role is to convert queries into queries/commands which are understandable for the specific data source. They convert query results from the source format to a format which is understandable for the application. Wrappers are useful (and enough) if the data represented by the different data models is structured.

4. Semi-structured Data

In this case we have data from sources that are not prescriptive, regular and complete like traditional DBMSs. We could have data from a xml file or from a web page.

Semi-structured data models are based on:

- Text
- Trees
- Graphs
    - Labelled nodes
    - Labelled arcs
    - Both

They are all different and do not lead to an easy integration.

In the case of semi-structured data, we would like to integrate, query and compare data with different structures also with structured data. So, an overall data representation should be progressively built as we discover and explore new information sources.

4.1 Mediators

A mediator is actor involved between the client and the data sources. It includes:

- the processing needed to make the interfaces work
- the knowledge structures that drive the transformations needed to transform data to information
- any intermediate storage that is needed (Wiederhold)

The problem is that we need a mediator for each type of semi-structured data in order to understand its semantics.

TSIMMIS

TSIMMIS is the first system based on the mediator/wrapper paradigm, proposed in the 90's at Stanford.

- unique, graph-based data model
    - OEM – Object Exchange Model, self-descriptive and each object appears in a nested structure)
- data model managed by the mediator
- wrappers for the model-to-model translations
- query posed to the mediator
    - the language is LOREL (Lightweight Object Repository Language)
- mediator knows the semantics of the application domain
    - each mediator is specialized into a certain domain thus each mediator must know domain metadata which convey the data semantics

4.2 Automatic wrappers

When a mediator is used, it is typically created by specializing it into a certain domain. Thus, each mediator must know domain metadata which convey the data semantics. If data source changes a little, the wrapper has to be modified. This is mostly because wrappers act based on extraction rules, which may be inappropriate in case their input changes.

This has led, in recent year, toward the research of the so-called automatic wrappers. Automatic wrappers are wrappers that may act in dynamic environments which, although changing, always present some regularities in the data. This situation typically occurs in data intensive websites.

Road Runner Project

The Road Runner Project is one of the most famous examples of automatic wrapper generation-based data integration process. It is based on the page class, a collection of pages generated by some script from a common dataset. It acts by finding the underlying dataset structure from a pool of sample HTML pages.

5. Semi-structured Data (2)

5.1 Ontologies

The use of ontologies is a way to solve the problem of automatic semantic matching. They are a formal and shared definition of a vocabulary of terms and their inter-relationships.

It has a grammar for using the terms to express something meaningful within a specified domain of interest. The vocabulary is used to express queries and assertions.

Ontological commitments are agreements to use the vocabulary in a consistent way for knowledge sharing.

The scope is to give a formal specification for using of a common vocabulary for automatic knowledge sharing. Formally specifying a conceptualization means giving a unique meaning to the terms that define the knowledge about a given domain.

There are two types of ontologies:

- Taxonomic ontologies
  - Definition of concepts through terms, their hierarchical organization, and additional (pre-defined) relationships (synonymy, composition, …)
  - To provide a reference vocabulary
- Descriptive ontologies
  - Definition of concepts through data structures and their inter-relationships
  - Provide information for "aligning" existing data structures or to design new, specialized ontologies (*domain ontologies*)
  - Closer to the database area techniques

An ontology consists of:

- concepts: generic concepts (express general world categories), specific concepts (describe a particular application domain, domain ontologies)
- concept definition (in natural language or via a formal language)
- relationships between concepts: taxonomies, user-defined associations, synonymies,...

Formal definition

A formal definition of an ontology identifies it as a quadruple (C, R, I, A), where:

• C is the set of concepts

• R is the set of relations between concepts

• A is the set of axioms

• I is the instance collection (e.g. the set of objects that are stored in the information source and belongs to a specific class or concept)

An ontology is (part of) a knowledge base, composed by:

- a T-Box, it contains all the concept and role definitions and all the axioms of our logical theory (e.g. “A father is a Man with a Child”).
- an A-box, it contains all the basic assertions (also known as ground facts) of the logical theory (e.g. “Tom is a father” is represented as Father(Tom)). It describes the instances.

OpenCyc
It is the open-source version of the Cyc technology born in 1984 at MCC.

The entire Cyc ontology containing hundreds of thousands of terms, along with millions of assertions relating the terms to each other, forming an ontology whose domain is all human consensus reality.

5.2 Semantic Web
The *semantic web* is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web.

Linked Data consists of connecting datasets across the Web. It describes a method of publishing structured data so that it can be interlinked and become more useful. It builds upon standard Web technologies such as HTTP, RDF and URIs, but extends them to share information in a way that can be read automatically by computers, enabling data from different sources to be connected and queried.

Open data is data uploaded to the Web and accessible to all.
Linked Open data extend the web with a data common by publishing various open datasets as RDF on the Web and by setting RDF links among them.

RDF
At the core of RDF is this notion of a triple subject-predicate-object, a statement that represents two vertices connected by an edge:

- Subject, a resource or a node in the graph
- Predicate, an edge / a relationship
- Object, another node or a literal value

The connection between a subject and an object is expressed in term of a predicate, which enforce a link with a specific meaning.

OWL
OWL is the first level above RDF. It is an ontology language that can formally describe the meaning of terminology used in Web documents, going beyond the basic semantics of RDF schema.

It is part of the growing stack of W3C recommendations related to the Semantic Web.

- **XML** provides a surface syntax for structured documents but imposes no semantic constraints on the meaning of these documents.

- **XML Schema** is a language for restricting the structure of XML documents and also extends XML with data types.
- **RDF** is a data model for objects ("resources") and relations between them, provides a simple semantics for this data model, and can be represented in an XML syntax.
- **RDF Schema** is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- **OWL** adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

OWL has three increasingly expressive sublanguages:

1. OWL Lite
2. OWL DL
3. OWL Full

5.3 Reasoning services for ontologies

Services for the Tbox

- Subsumption: verifies if a concept C subsumes (is a sub concept of) another concept D
- Consistency: verifies that there exists at least one interpretation I which satisfies the given Tbox
- Local Satisfiability: verifies, for a given concept C, that there exists at least one interpretation in which C is true.

Services for the Abox

- Consistency: verifies that an Abox is consistent with respect to a given Tbox
- Instance Checking: verifies if a given individual x belongs to a particular concept C
- Instance Retrieval: returns the extension of a given concept C, that is, the set of individuals belonging to C.

5.4 Ontologies and integration problems

A database and an ontology are not distant concepts: they serve, instead, different purposes. It is possible to see that an ontology may be translated to an ER and viceversa, but this implies also a degradation in terms of the properties of the other.

How should we improve database conceptual models to fulfil ontology requirements?

- Supporting defined concepts and adding the necessary reasoning mechanisms
- Managing missing and incomplete information; semantic differences between the two assumptions made w.r.t. missing information (Closed World Assumption vs. Open World Assumption)

There are some problems:

- Discovery of equivalent concepts (mapping)
- Formal representation of these mappings

- Reasoning of these mappings

Ontology matching
It is the process of finding pairs of resources coming from different ontologies which can be considered equal in meaning – matching operators but we need some kind of similarity measure.

Ontology mapping
It is the process of relating similar concepts or relations of two or more information sources using equivalence relations or order relations.
These relations are commonly implemented in inference and reasoning software's, so we can use the output ontology to perform complex tasks on them without extra effort.

Reasons for ontology mismatches

- Scope: Two classes seem to represent the same concept, but do not have exactly the same instances
- Model coverage and granularity: a mismatch in the part of the domain that is covered by the ontology, or the level of detail to which that domain is modelled.
- Paradigm: Different paradigms can be used to represent concepts such as time. For example, one model might use temporal representations based on continuous intervals while another might use a representation based on discrete sets of time points.
- Encoding
- Concept description: e.g. a distinction between two classes can be modelled using a qualifying attribute or by introducing a separate class, or the way in which is-a hierarchy is built
- Homonymies
- Synonymies

5.5 How can ontologies support integration?
An ontology may, in principle, serves as an integration support tool. The type of aid an ontology may represent is divided in four classes:

1. An ontology can be a schema integration support tool. Ontologies are used to represent the semantics of schema elements (if the schema exists)
2. An ontology can be used instead of a global schema:
    a. schema-level representation only in terms of ontologies
    b. ontology mapping, merging etc. instead of schema integration
    c. integrated ontology used as a schema for querying
    Data source heterogeneity is solved by extracting the semantics in an ontological format.
    Data source ontologies are mapped to the Domain Ontology (Global Schema). This allows to solve problem like impedance mismatch and unstructured data source management.
3. An ontology as a support for content interpretation and wrapping (e.g., HTML pages).
4. An ontology as a mediation support tool for content inconsistency detection and resolution (record linkage and data fusion).

Also ontologies require query languages for:

- Schema exploration
- Reasoning on the schema
- Instance querying

An example of ontology query language is *SPARQL*.

When we use ontologies to interact with databases, we have to take care of:

- Transformation of ontological query into the language of the data source, and the other way round
- Different semantics (CWA versus OWA)
- What has to be processed where (e.g. push of the relational operators to the relational engine)

6. Introduction to data warehouse

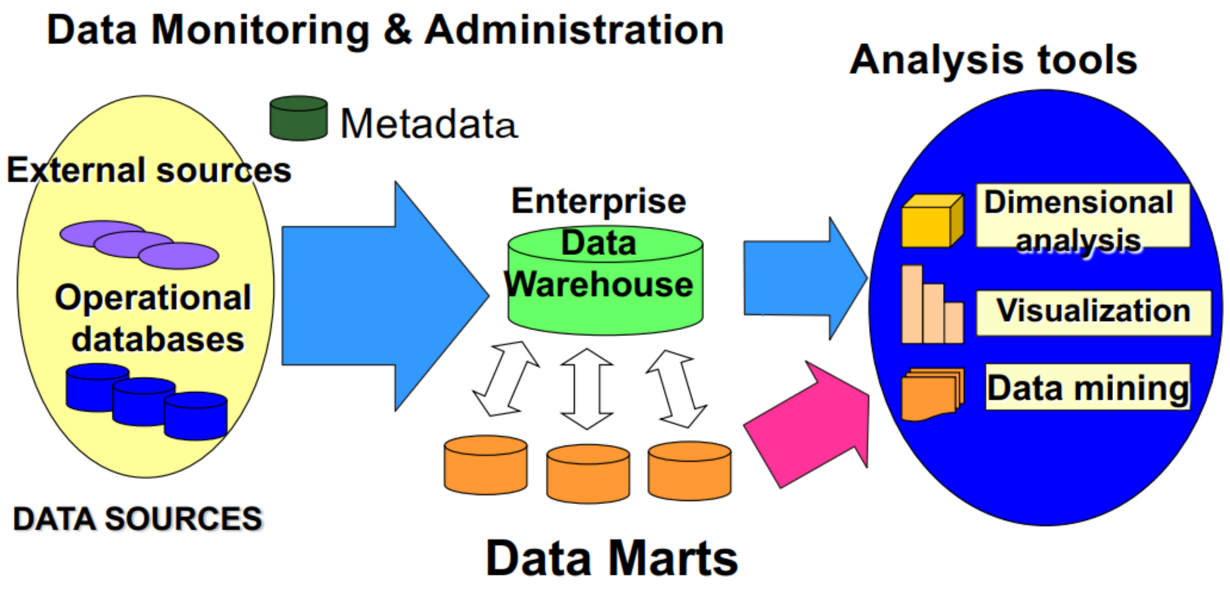
A data warehouse is a *subject-oriented*, *integrated*, *time-varying*, *non-volatile* collection of data that is used primarily in organizational decision making.

The main purpose of a data warehouse is to allow systematic or ad-hoc data analysis and mining. They are very big in terms of space occupied and they differ from "*data lakes*" (or dataspaces).

Data warehouse	Dataspace (or data lake)
Usually Relational DBMS	HDFS/Hadoop
Data of recognized high value	Candidate data of potential value
Aggregated data	Detailed source data
Data conforms to enterprise standards	Fidelity to original format and condition (transformed/cleaned later, on demand, when needed)
Known entities	Raw material (metadata) to discover entities
Data integration up front	Data integration on demand
Schema on write	Schema on read (when doing analytics)

As written above, a data warehouse is stored in a relational DBMS, but it is a specialized DB with different kind of operation. We have mostly reads, long queries, lots of scans, summarized data and hundreds of users working on at the same moment.

Data warehouse is useful in commerce, manufacturing plants, financial services, telecommunications, etc.



6.1 Dimensional Fact Model (DFM)

We can imagine an insurance company with a data warehouse built like a cube where we have three dimensions, year, age, type of insurance, and the metric values are our data like num. of policies.

The DFM allows one to describe a set of fact schemata. The components are:

- Facts
    - It is a concept that is relevant for the decisional process; typically it models a set of events of the organization
- Measures
    - It is a numerical property of a fact
- Dimensions
    - It is a fact property defined with regards to a finite domain
- Dimension hierarchy
    - Like day → month → trimester → year

In the example explained above, the fact could be "*policies*".

In this way we have a **multidimensional view** of the data.

6.2 OLAP operations

- roll-up
    - Aggregates data at a higher level – e.g. last year's sales volume per product category and per region
- drill-down
    - De-aggregates data at the lower level – e.g. for a given product category and a given region, show daily sales
- slice-and-dice
    - Applies selections and projections, which reduce data dimensionality
- pivoting
    - Selects two dimensions to re-aggregate data (cube reorientation)
- ranking
    - Sorts data according to predefined criteria
- traditional operations (select, project, join, derived attributes, etc.)

We have two different OLAP logical models:

- **MOLAP** (Multidimensional On-Line Analytical Processing), it stores data by using a multidimensional data structure (a real data cube)
- **ROLAP** (Relational On-Line Analytical Processing, it uses the relational data model to represent multidimensional data (the most used)
    - In the ROLAP model, there is the use of a new polymorphic value, "*ALL*". It expresses all the possible tuple aggregations of a table when we do a query.

In order to integrate similar aggregation capabilities in SQL (for supporting ROLAP operations), two instructions were defined:

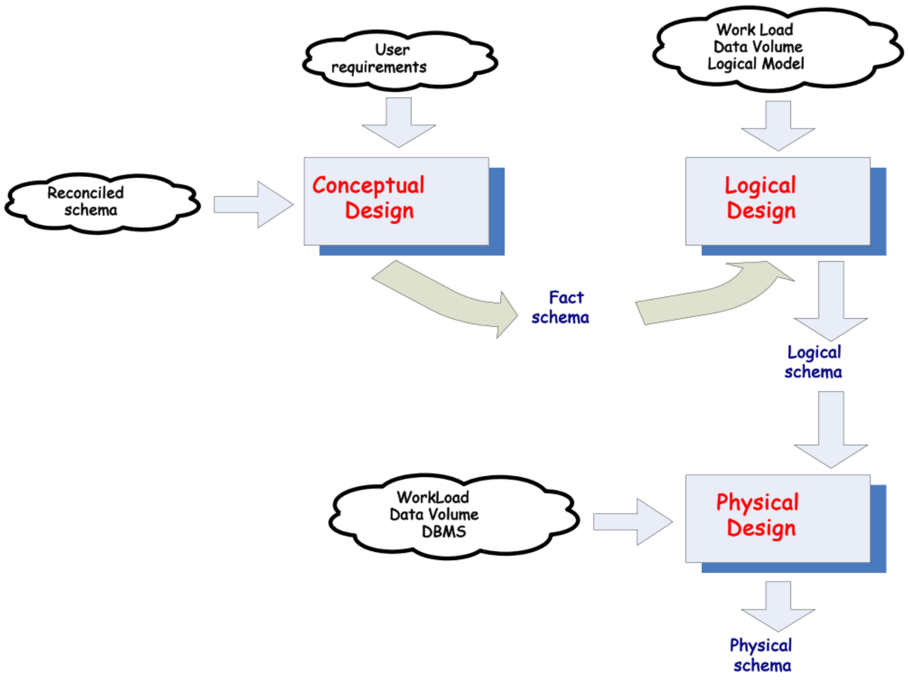
- **WITH CUBE**
    - generates a result set that shows aggregates for all combinations of values in the selected columns

    - evaluates aggregate expression with all possible combinations of columns specified in group by clause

- **WITH ROLLUP**
    - generates a result set that shows aggregates for a hierarchy of values in the selected columns
    - evaluates aggregate expressions only relative to the order of columns specified in group by clause
    - it eliminates the results that contain ALL only in one column

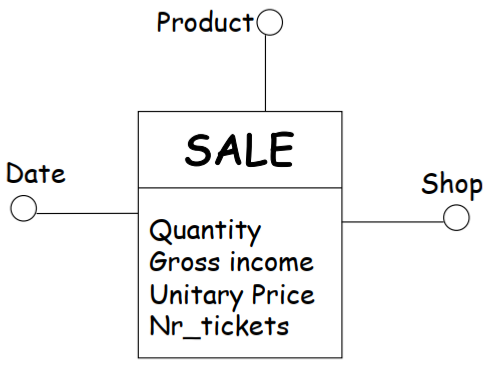
7. Data warehouse design

Data Warehouses are based on the multidimensional model. A common conceptual model for DW does not exist, in fact, the Entity/Relationship model cannot be used in the DW conceptual design.



7.1 Conceptual model

Fact schema



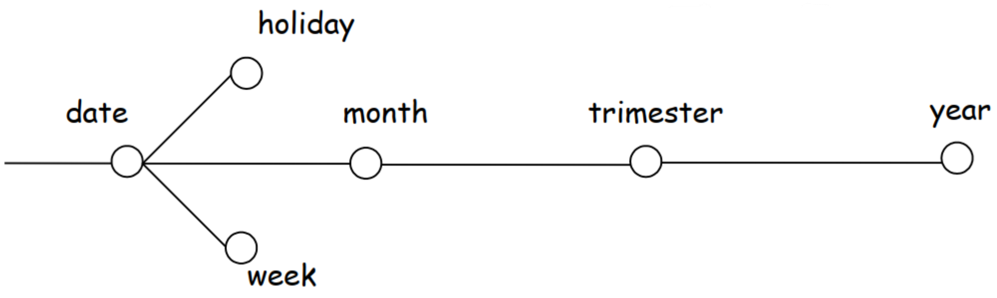
A **fact** describes an N:M relationship among its dimensions. There must be a functional dependency between the fact and its dimensions.
Naming convention: the dimensions of a same fact schema must have distinct names.

Fact → SALE
Measures → Quantity, Gross income, etc.
Dimensions → Date, Product, Shop

A **dimensional attribute** must assume discrete values, so that it can contribute to represent a dimension. Dimensional attributes can be organized into **hierarchies**. They (hierarchies) describe how it is possible to group and select primary events, in fact, the root of a hierarchy corresponds to

the primary event and represents the *finest aggregation granularity*.
A **dimensional hierarchy** is a directional tree where:

- Nodes are dimensional attributes
- Edges describe N:1 relationship between pairs of dimensional attributes
- Root is the dimension



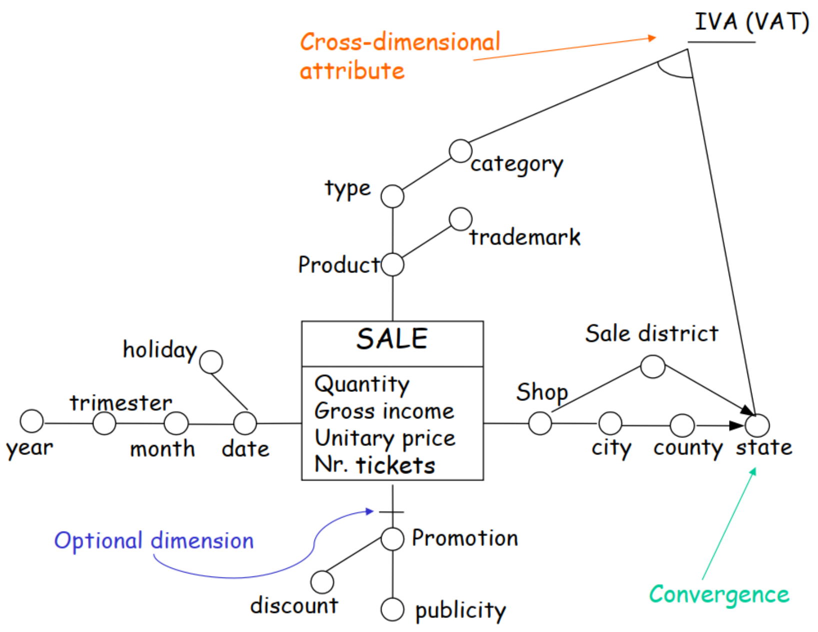
A **primary event** is an occurrence of a fact.
A **secondary event** aggregates some of the corresponding primary events, for example, when we use "group by" in a query.

In the fact schema we can have **descriptive attributes**. They contain additional information about the linked dimensional attribute. In our case we have "shop", a descriptive attribute can be the "address" of a specific shop.

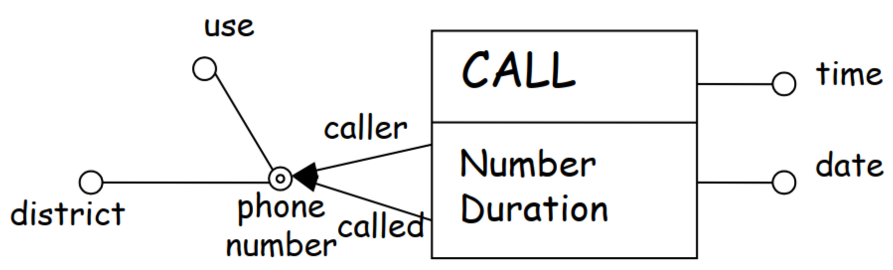
There could be also **optional edges** and **optional dimensions** that are not required but can be added to more clear/precise.

A **cross-dimensional attribute** is a dimensional or a descriptive attribute whose value is obtained by combining values of some dimensional attributes.

In the fact schema we can have **convergence**; this means that two dimensional attributes may be connected by more than two distinct directed edges.



In a fact schema, some portions of a hierarchy might be duplicated, so we allow **hierarchy sharing**. If the sharing starts with a dimension attribute, it is necessary to indicate the roles on the incoming edges.
Necessary condition: the functional dependency must hold on to both branches.



As we said before, there must be a functional dependency between the fact and each dimension. Some attributes, or some dimensions, may be related by a many-to-many relationship and we denote them by using **multiple edges**.

In the fact schema we have to specify the **aggregability** when a measure is non-additive with regards to a specific dimension.
(e.g., imagine having a fact, “Inventory”, and a measure “Q.ty in stock”, we have to specify that it is non-additive with regards to the time dimension linked to the fact)
Aggregation requires to specify an operator to combine the values related to primary events into a unique value related to a secondary event (e.g., sum of sold quantity aggregated by month).

A fact schema is **empty** if there are no measures. In fact, the default measure is the **count**.

7.2 Conceptual design

Conceptual design considers the documentation related to the integrated database.

Top-down methodology

1. Fact choice
2. For each fact:
    a. Design of the attribute tree
    b. Attribute-tree editing
    c. Dimension definition
    d. Measure definition
    e. Fact schema creation

The **starting point** is the Relational and/or the E/R schema. Then, we have to choose a fact or more than one. They correspond to events that dynamically happen in the organization.
In a E/R schema, a fact can correspond to an entity (or table) or to a relationship. A good candidate is an entity or a relationship that is updated frequently.

Attribute tree
It is composed by:

- Nodes, corresponding to attributes (simple or complex) of the source schema
- Root, corresponding to the primary key of the fact we analyse

- For each node, the corresponding attribute functionally determines its descendant attributes

After the creation of the attribute tree, we have to edit it in order to remove some attributes which are irrelevant. We can perform two operations:

- **Pruning** of a node *v*: the subtree rooted in *v* is deleted
- **Grafting** of a node *v*: the children of *v* are directly connected to the father of *v*

There also **special cases** like when we have cyclic relationships or cycles in the schema.
In the first case, they must be broken after a certain number of iterations.
In the second case, they must be broken keeping the most convenient link.

Dimension definition
Dimensions can be chosen among the attributes that are children of the root of the tree.
(Time should always be a dimension)

Measure definition
While the set of attributes that acts as fact identifier is included in the set of dimensions, numerical attributes of the root (fact) are measures (e.g., unitary price)

More measures are defined by applying aggregate functions to other attributes of the tree (e.g., ticket count). Typical aggregate functions: sum, average, min, max, count.

Glossary
In the glossary, an expression is associated with each measure. The expression describes how we obtain the measure at the different levels of aggregation starting from the attributes of the source schema.

Fact schema creation
At this point the attribute tree is translated into a fact schema including dimensions and measures.

7.3 Logical design

Data Mart logical models

- **MOLAP** (Multidimensional On-Line Analytical Processing) stores data by using a multidimensional data structure
    - It is the more traditional way of OLAP analysis. Data is natively stored in a multidimensional cube. The storage is not in the relational database, but in proprietary formats.
    - **Advantages**: excellent performance. MOLAP cubes are built for fast data retrieval and are optimal for slicing and dicing operations. Can perform complex calculations: All calculations have been pre-generated when the cube is created. Hence, complex calculations are not only doable, but they return quickly.
    - **Disadvantages**: limited in the amount of data it can handle. Because all calculations are performed when the cube is built, it is not possible to include a large amount of data in the cube itself. This is not to say that the data in the cube cannot be derived from a large amount of data. Indeed, this is possible, but in this case, only summary-level information will be included in the cube itself.

Requires additional investment: Cube technologies are often proprietary and do not already exist in the organization. Therefore, to adopt MOLAP technology, chances are additional investments in human and capital resources are needed.

- **ROLAP** (Relational On-Line Analytical Processing) uses the relational data model to represent multidimensional data
    - It relies on manipulating the data stored in the relational database to give the appearance of traditional OLAP's slicing and dicing functionality. In essence, each action of slicing and dicing is equivalent to adding a "WHERE" clause in the SQL statement.
    - **Advantages**: Can handle large amounts of data. The data size limitation of ROLAP technology is the limitation on data size of the underlying relational database. Can leverage functionalities inherent in the relational databases.
    - **Disadvantages**: Performance can be slow. Because each ROLAP report is essentially a SQL query (or multiple SQL queries) in the relational database, the query time can be long if the underlying data size is large.
      Limited by SQL functionalities: Because ROLAP technology mainly relies on generating SQL statements to query the relational database, and SQL statements do not fit all needs (for example, it is difficult to perform complex calculations using SQL), ROLAP technologies are therefore traditionally limited by what SQL can do. ROLAP vendors have mitigated this risk by building into the tool out-of-the-box complex functions as well as the ability to allow users to define their own functions.
- **HOLAP** (Hybrid Holap): attempts to combine the advantages of MOLAP and ROLAP.
  For summary-type information, HOLAP leverages cube technology for faster performance. When detail information is needed, HOLAP can "drill through" from the cube into the underlying relational data.

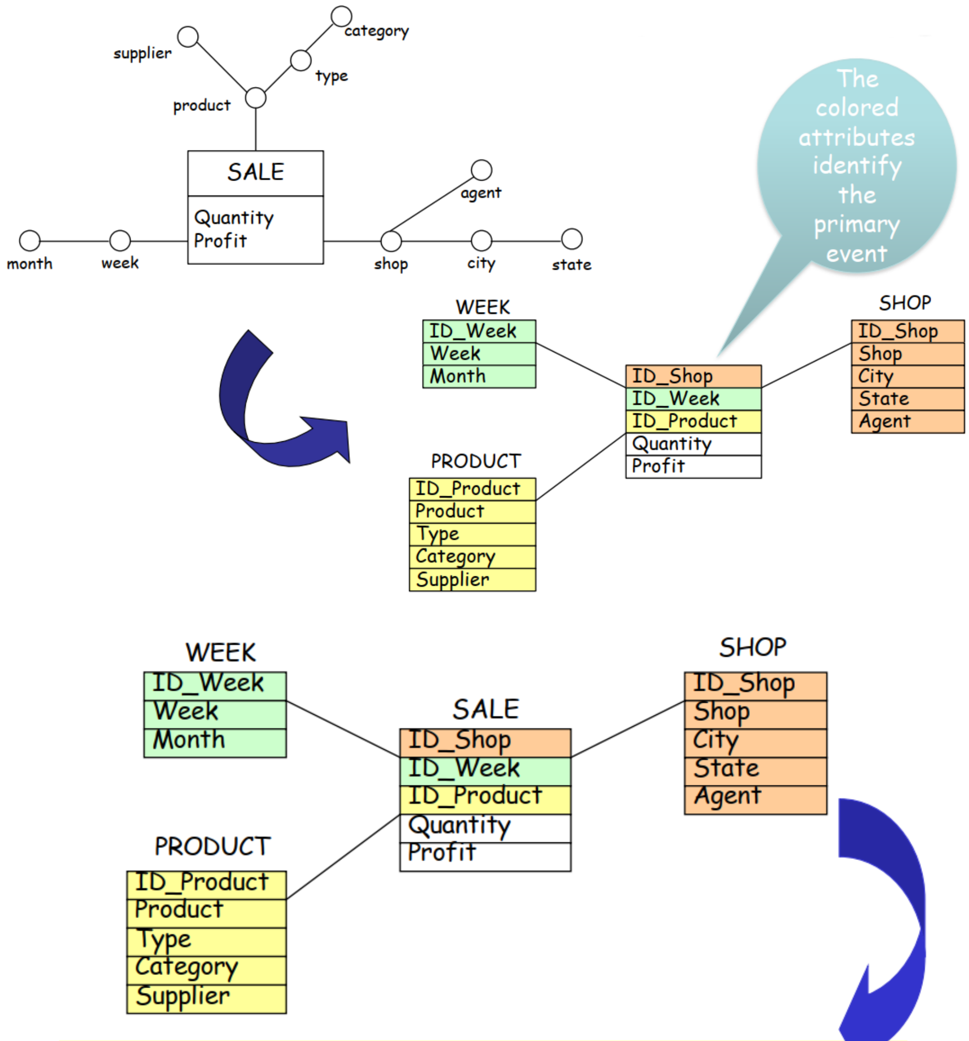
Star Schema

It is based on the Star Schema. A star schema is :

- A set of relations $DT_1, DT_2, \dots, DT_n$ - dimension tables - each corresponding to a dimension.
- Each DT_i is characterized by a primary key $d_i$and by a set of attributes describing the analysis dimensions with different aggregation levels
- A relation FT, fact table, that imports the primary keys of dimensions tables. The primary key of FT is $d_1\ d_2 \dots d_n$; FT also contains an attribute for each measure.

Some considerations on this schema:

- The fact table contains information expressed at different aggregation levels.
- Dimension table keys are surrogates (i.e. generated ids), for space efficiency reasons.
- Dimension tables are de-normalized: note that $product \ \rightarrow type \rightarrow category$ is a transitive dependency.
- De-normalization introduces redundancy, but fewer joins to do.



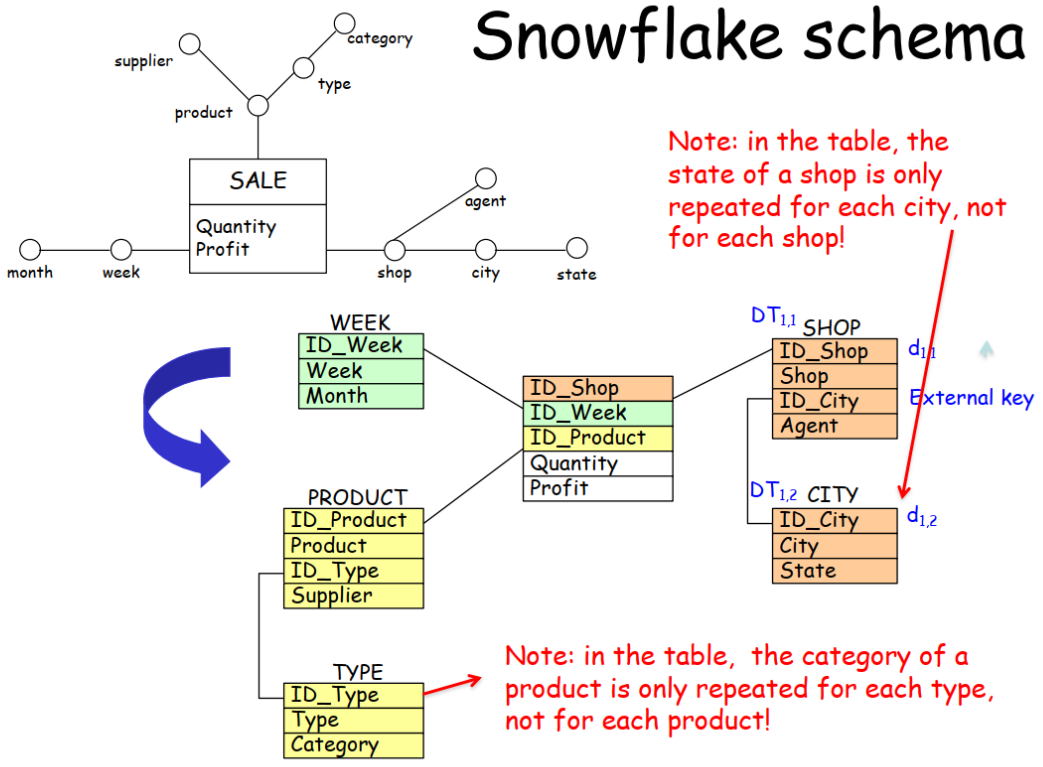
```
select City, Week, Type, sum(Quantity)
from   Week, Shop, Product, Sale
where  Week.ID_Week=Sale.ID_Week and
       Shop.ID_Shop=Sale.ID_Shop and
       Product.ID_Product=Sale.ID_Product and
       Product.Category = 'FoodStuff'
group by City,Week,Type
```

Snowflake Schema

The snowflake schema reduces the denormalization of the dimensional tables DT_i of a star schema. It removes some of those transitive dependencies.
Dimensions tables of a snowflake schema are composed by:

- A primary key $d_{i,j}$
- A subset of DT_i attributes that directly depend on $d_{i,j}$
- Zero or more external keys that allow to obtain the entire information

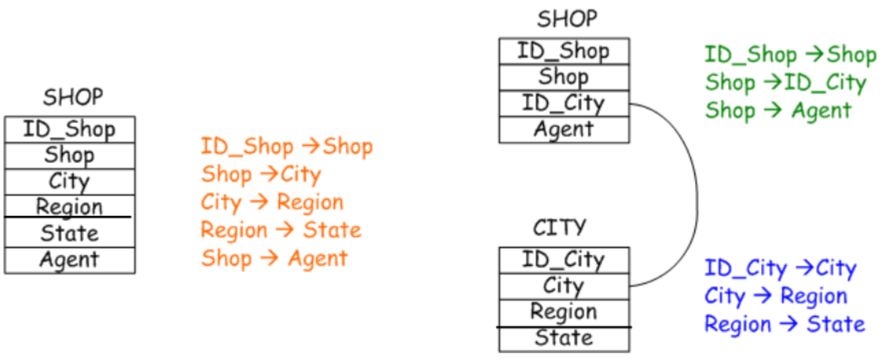
Snowflake schema



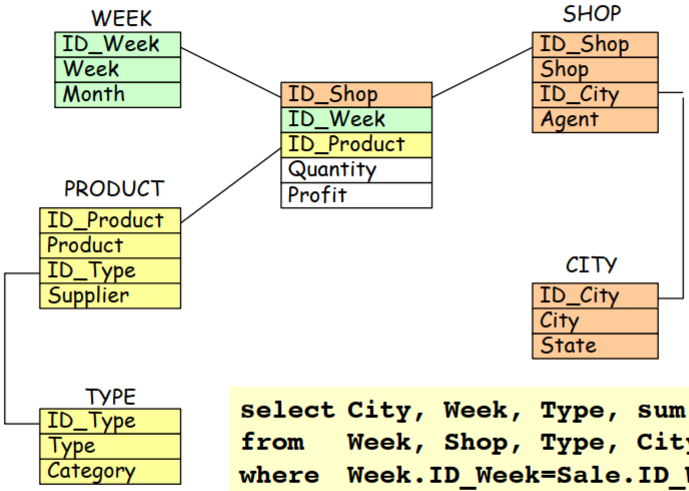
Some considerations:

- Reduction of memory space
- New surrogate keys
- Advantages in the execution of queries related to attributes contained into fact and primary dimension tables

If there exists a cascade of transitive dependencies, attributes depending (transitively or not) on the snowflake attribute are placed in a new relation.



OLAP queries on snowflake schema



```
select City, Week, Type, sum(Quantity)
from   Week, Shop, Type, City, Product, Sale
where  Week.ID_Week=Sale.ID_Week and
       Shop.ID_Shop=Sale.ID_Shop and
       Shop.ID_City = City.ID_City and
       Product.ID_Product=Sale.ID_Product and
       Product.ID_Type=Type.ID_Type and
       Product.Category = 'FoodStuff'
group by City,Week, Type
```

Views

Aggregation allows to consider concise (summarized) information. This operation is very expensive, so we need a pre-computation. A **view** denotes a fact table containing aggregate data and this allows us to perform that per-computation.

A view can be characterized by its aggregation level (pattern):

- Primary views: correspond to the primary aggregation levels
- Secondary views: correspond to secondary aggregation levels (secondary events)

Sometimes it is useful to introduce new measures in order to manage aggregations correctly. Derived measures can be obtained by applying mathematical operators to two or more values of the same tuple. These operators can be:

- Distributive operator, allows to aggregate data starting from partially aggregated data (e.g., sum, max, min)
- Algebraic operator, requires further information to aggregate data (e.g., avg)
- Holistic operator, it is not possible to obtain aggregate data starting from partially aggregate data (e.g., mode, median)

Currently, aggregate navigators are included in the commercial DW system. They allow to re-formulate OLAP queries on the "best" view. They manage aggregates only by means of distributive operators.

(stop at slides n. 24 – pdf 10)

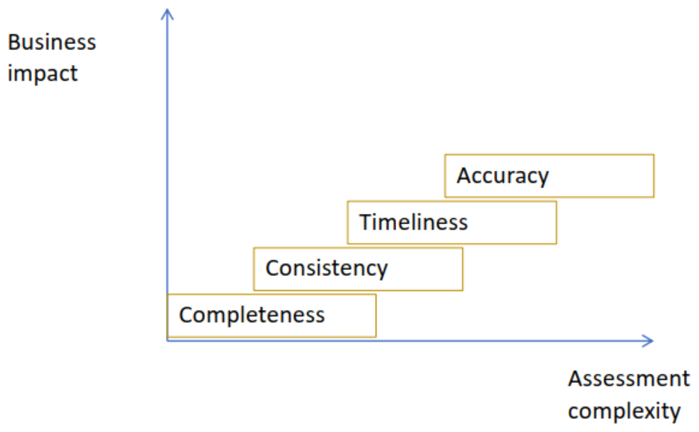
8. Data Quality

Data quality refers to the accuracy, completeness, reliability, and relevance of data. It is a measure of how well data serves its intended purpose in a specific context. High-quality data is essential for making informed decisions, conducting accurate analyses, and achieving meaningful insights.

These analyses have brought many companies to pursue a new type of business management approach called Data-Driven Management. It is the management of the business decision derived from these analyses. This leads to the fact that DDM and Data Quality are strictly related since companies want to be accurate and efficient in the business decision making. The quality of the data analysed is crucial.

Most common dimensions in Data Quality are:

- Accuracy
    - Refers to the correctness of data. Accurate data is free from errors and discrepancies.
- Completeness
    - Involves the extent to which data is whole and contains all the necessary information. Incomplete data may lack certain values or details.
- Consistency
    - Addresses the uniformity of data across different databases, systems, or sources. Consistent data should not contradict itself.
- Timeliness
    - Reflects how up to date the data is. Timely data is relevant and current for the intended use.



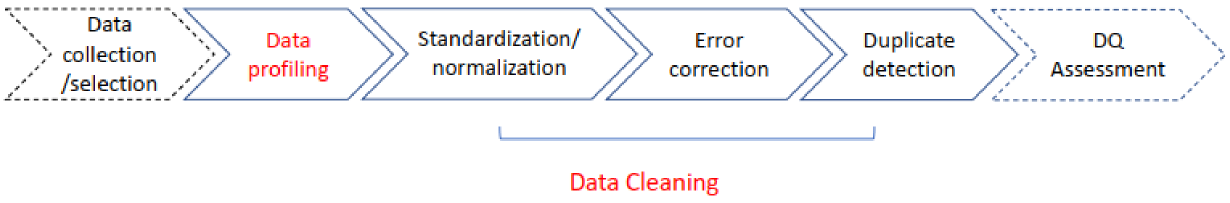
Of course, companies can improve their data quality following some strategies. There are:

- Data-based approaches
    - They focus on data values and aim to identify and correct errors without considering the process and context in which they will be used
- Process-based approaches

- They are activated when an error occurs and aim to discover and eliminate the root cause of the error

8.1 Data cleaning

Data cleaning is the process of identifying and eliminating inconsistencies, discrepancies, and errors in data in order to improve quality.



A crucial step in this process is the second one, **Data profiling**. The goal of data profiling in data quality is to provide a comprehensive understanding of the dataset, reveal potential issues, and guide subsequent data cleaning and improvement efforts. In the data management lifecycle it contributes to the overall effectiveness and trustworthiness of the data used for decision-making and analysis.

Data transformation and normalization

- Data type conversion
- Mapping data into a common format
- Discretization of numerical values
- Domain-specific transformation

Error localization and correction

- Localization and correction of inconsistencies
- Localize and correction of incomplete data
- Localization of outliers
    - “suspicious” observation that deviates too much from other observations. An outlier is then a value that is unusually larger or smaller in relation to other values in a set of data

Duplicate detection

Duplicate detection (or entity reconciliation) is the discovery of multiple representations of the same real-world object.

9. Data ethics

Ethics plays a crucial role in data management for several reasons, reflecting the need to balance the benefits of data usage with the potential risks and societal implications. With search and recommendation engines, the web can largely influence our lives, e.g. by recommending interesting jobs only to white males, discriminating as an effect of biased data or algorithms. With statistics used everywhere, it may happen that very critical decisions be taken without taking their ethical consequences into account.

The illusion is that Big Data processing is based on algorithms that must be objective, but this is not true because these algorithms are based on data, and data may contain bias. For this reason technology must be accompanied by *ethical* and *legal* considerations and competence.

It is up to the data scientists to identify which datasets can genuinely help answering some given question, understand their contents and choose the most appropriate knowledge extraction technique (search, query, or data analysis/AI methods) to obtain a fair result.
This sequence of choices may strongly influence the process, and biased results might be obtained.

Ethical dimensions:

- Data protection
    - Traditional knowledge extraction systems, including database systems, search engines, data warehouses, etc., hardly pay specific attention to ethically sensitive aspects of knowledge extraction processes and their outcomes. Such aspects are now becoming prominent, especially with regard to the protection of human rights and their consequences in normative ethics. These demands are broadly reflected into codes of ethics for companies and computer professionals, and also in legally binding regulations such as the EU General Data Protection Regulation (GDPR)
- Fairness → fairness of data is defined as the lack of bias
    - Various "categories" of fairness have been identified, like equality vs equity
- Diversity
    - It is the degree to which different kinds of objects are represented in a dataset. Several metrics of diversity are proposed.
        - Ensuring diversity at the beginning of the information extraction process may be useful for enforcing fairness at the end.
        - The diversity dimension may conflict with data quality needs, that prioritize the use of few, high-reputation sources.
- Transparency
    - It is the ability to interpret the information extraction process in order to verify which aspects of the data determine its results. In this context, transparency metrics can use the notions of
        - data provenance, describing where the original data come from
        - explanation, describing how a result has been obtained
    - Transparency may conflict with Data Protection

Data quality is a typical **ethical requirement**: we could never trust a piece of information if it did not have the typical data quality properties. Yet, we can also assert the opposite: that data should conform to a high ethical standard, for it to be considered of good quality. Hence, the satisfaction of the ethical requirements is necessary to assert the quality of a result. It is the responsibility of the system designer, and of the person/company that ordered the job, to ensure that the (necessary) ethical properties are satisfied.

10. New trends in Data Integration

10.1 Uncertainty in Data Integration

Databases are assumed to represent certain data (a tuple in the database is true). But real life is not as certain. Uncertain databases attempt to model uncertain data and to answer queries in an uncertain world. Moreover:

- Data itself may be uncertain (e.g. extracted from an unreliable source)
- Mappings might be approximate
- Reconciliation is approximate
- Imprecise queries are approximate

Whatever the semantics of uncertainty (e.g., fuzzy or probabilistic...) an uncertain database describes a set of possible worlds.

Example: assign each tuple a probability. Then the probability of a possible world is the product of the probabilities for the tuples.

10.1.1 Entity Resolution

Entity Resolution (ER) is the problem of accurately identifying multiple, differing, and possibly contradicting representations of unique real-world entities in data.

The causes of multiple representations of the same real-world entity are:

- Noise (small errors)
- Inconsistencies (conflicting or different values for properties/relations of an object The record matching rules are used to detect duplicates in data.

A common approach is to use **record-matching rules**. Record-matching rules are constraints that state: *"if any two records are similar on properties P1, ... ,Pn, then they refer to the same entity".*

Problems of this approach:

- it is difficult to generate automatically record-matching rules
- one-to-one record matching is too expensive (quadratic)

10.1.2 Data Provenance

Sometimes knowing where the data have come from and how they were produced is critical (e.g., an information extractor might be unreliable, or one data source is more authoritative than others). **Provenance of a data** item records "where it comes from":

- Who created it
- When it was created
- How it was created (as value in a database, as the result of a computation, coming from sensor, etc ...)

Two viewpoints on Provenance (even though they are perfectly equivalent):

- **Provenance as annotations on data**: models' provenance as a series of annotations describing how each data item was produced. These annotations can be associated with tuples or values
- **Provenance as a graph of data relationships**: models' provenance as a graph, with tuples as vertices. Each possible direct derivation of a tuple from a set of source tuples is a edge-connecting the source and derived tuples.

Provenance is achieved via various techniques. In principle, determining provenance of data requires, giving a suitable explanation of the nature of the data, scoring the source and the data for assessing its quality and evaluating the influence of a source with respect to another.
These steps may be complex to perform manually; at the same time, they may not be automated. **Crowdsourcing** provides a cheap and scalable way of annotating provenance of data; a set of people is rewarded to annotate provenance of data.

10.2 Building Large-Scale Structured Web Databases

Building a large-scale structured web database is a subtle task. It poses new problems with respect to the one discussed, which may be summarised in this objective: develop methodologies for designing a fully integrated database coming from heterogeneous data sources. However, depending on the type of system we aim to integrate, we may also simplify the steps required for dealing with these problems.

10.2.1 Lightweight Integration

We may need to integrate data from multiple sources to answer a question asked once or twice. The integration must be done quickly and by people without technical expertise. This problem is defined ad **lightweight integration**, as it is a small-scale integration process with prioritise speed over "stability" (a solid and long-lasting implementation). Problems typical of lightweight data integration:

- Locating relevant data sources
- Assessing source quality
- Helping the user understand the semantics
- Support the process of integration

10.2.2 Mashup

Mashup is a paradigm for lightweight integration. It is an application that integrates two or more mashup components at any of the application layers (data, application logic, presentation layer) possibly putting them in communication with each other.

Key elements:

- **Mashup component**: is any piece of data, application logic and/or user interface that can be reused and that is accessible either locally or remotely (e.g., Craiglist and GoogleMaps)
- **Mashup logic**: is the internal logic of operation of a mashup; it specifies the invocation of components, the control flow, the data flow, the data transformations, and the UI of the mashup.

Mashups introduce integration at the presentation layer and typically focus on non-mission-critical applications.

Benefits

- Easy development of situational applications for power users
- Fast prototyping for developers
- Increased ROI for Service-Oriented-Application investments

Problems

- Mashup development is complex, due to the need of integrating different elements from different sources.

Luckily, mashups typically work on the ”surface”.

- Reuse of existing components
- Composition of the outputs of software systems

The work of developers can be facilitated by suitable abstractions, component technologies, development paradigms and enabling tools.

A mashup generally differs from the other integration practices as it also lies on the logic and presentation layer (it applies integration also with respect on *how* the user perceive the informative contribute of different sources), focusing on non-mission-critical application. Consider, in fact, that a mashup *may not be data related*!

Types of mashups

- **Data mashups**: retrieve data from different resources, process them and return an integrated result set. Data mashups are a Web-based, lightweight form of data integration, intended to solve different problems.
- **Logic mashups**: integrate functionality published by logic or data components. The output is a process that manages the components, in turn published as a logical component.
- **User interface mashups**: combine the component’s native UIs into an integrated UI. The output is a Web application the user can interact with. It is mostly client-side, generally short-lived.
- **Hybrid mashups**: span multiple layers of the application stack, bringing together different types of components inside one and a same application.

Data mashups are the lightweight form of data integration, which serve a different purpose with respect to normal data integration techniques: it is specific for ad-hoc data analyses, it’s simple and not suitable for critical tasks.

10.2.3 Pay-as-you-go management

In contrast the other kind of data integration, **pay-as-you-go systems** try to avoid the need for the initial setup phase that includes creating the mediated schema and source descriptions. The goal is to offer a system, that provides useful services on a collection of heterogeneous data with very little initial effort.

10.3 Data Spaces

A Database Management System (DBMS) is a generic repository for the storage and querying of structured data. Unfortunately in data management scenarios today it is rarely the case that all the data can be fit nicely into a conventional relational DBMS, or into any other single data model or system. Moreover, they often require a *semantic integration* phase beforehand, thus demanding additional work.

One solution is using a **dataspace**: it does not represent a data integration approach, rather, it is more of a data coexistence approach. The goal of dataspace support is to provide base functionality over all data sources, regardless of how integrated they are. Moreover, this is simplified as a dataspace tries to leverage pre-existing mapping and matching generation techniques.

Note that, although Dataspace represents a valid solution to the problem stated before, it does not guarantee the same benefits of a classical integration system. In fact, it does not fully enjoy of durability and consistency of data.

10.3.1 Data Lakes

The **Data Lake** represents a practical application of the concept of dataspace. A data lake is a big collection of (mostly unstructured) data which is stored exclusively in a raw format. In contrast to a database, a data lake does not rely on a schema, and accepts a great variety of data types. Moreover, it lacks ETL processes.

A data lake presents a series of advantages over a traditional database:

- It allows to deal with big-scale data volumes, at a relatively small cost
- Preserving the data ”as it is” allow to fully preserve its analysis potential, favouring machine learning purposes (profiling, analytics...)
- It allows wider opportunities of ”specialisation”, as data in its original format may furtherly be processed for further needs

However, one should also note the great disadvantage of data lakes:

- Lack of organisation in terms of data annotation and processing
- Redundancies in data
- No consistency guarantees
- Stricter requirements in terms of computational power when searching for a specific information (scan required)
- Requires processing method when data should be used

Although presenting problems, data lakes are, in principle, dataspaces: therefore, the disadvantages presented may be mitigated by on demand data integration techniques.